INFORMATION EXTRACTION AND SPEECH RECOGNITION

Ralph Grishman

Computer Science Department
New York University
New York, NY 10003

ABSTRACT

Information extraction is the process of analyzing natural language and collecting information about specified types of entities, relationships, or events. This paper provides an overview of a range of information extraction tasks, and briefly describes the structure of systems for performing these tasks. The paper then considers the impact of speech recognition errors on such systems, and describes some experiments which indicate how the effect of such errors may be reduced. Finally, the paper speculates on avenues of research which may enhance the performance of extraction systems on speech transcripts.

WHAT IS INFORMATION EXTRACTION?

Information extraction systems identify particular types of entities, relations, or events in natural language text. In some cases the result is a marked-up text (say, with some entities highlighted); in other cases, it is a data base containing the critical pieces of information about each event or relation (such as the participants, date, location, effect, etc.). Experimentation with extraction systems has been growing rapidly, reflecting both the need to process increasing amounts of machine-readable text (such as on-line newspapers) and the ability of computational linguists to construct relatively robust extraction systems.

Of course, "identifying entities, relations, or events" can cover a wide range of systems. In the United States, the notion of what constitutes information extraction has been heavily influenced by the Message Understanding Conferences (MUCs) (Grishman and Sundheim 96, MUC 95). The focus of these Conferences, like DARPA Speech Workshops, has been on the evaluations which preceded them. For most of the MUCs, the evaluation involved a single task of event extraction, but recent MUCs have added a range of evaluation tasks. The current MUC evaluation (MUC-7) includes five tasks applied to English text: named entities, template elements, template relations, coreference, and scenario templates.[1] We shall look at these tasks (except for coreference) briefly.

Named Entity

The four tasks we shall examine form a hierarchy, with the named entity task the simplest, the template entity task building on the efforts of named entity recognition, and the template relation and scenario template tasks building on the results of the template entity task.

The named entity task primarily involves identifying and classifying names in text. Specifically, names referring to individual people, to organizations, and to locations are to be tagged. The text is tagged using SGML-style markup, using a tag of "ENAMEX" with TYPE="PERSON", "ORGANIZATION", or "LOCATION". A few other structures are also to be tagged: date expressions ("January 10", "last week"), time expressions ("4 PM", "noon"), currency ("$10 million"), and percentages ("10%"). A sample named entity annotation is shown in Figure 1.

```
Mr. <ENAMEX TYPE="PERSON">Sears</ENAMEX>
bought a new suit in
<ENAMEX TYPE="ORGANIZATION">Sears</ENAMEX>
in
<ENAMEX TYPE="LOCATION">Washington</ENAMEX>
 <TIMEX TYPE="DATE">yesterday</TIMEX>.
```

Figure 1. Named Entity annotation

Template Element

A person or organization may be referred to several times in an article, sometimes by names in different forms, sometimes by descriptions. For example, an article might mention "Carol Mantix" and later refer to her as "Ms. C. Mantix" and "that well-know raconteur and computational linguist". In the template element task, we create a form of data base for each article, with one entry for each person and organization mentioned in the article.[2] The entry lists all the forms (names and descriptions) which have been used to refer to the entity in the article. A sample entry is shown in Figure 2.

[1] In addition, the parallel MET evaluation is performing multilingual named entity extraction — for Japanese, Chinese, and Thai.

[2] MUC-7 also added template elements for a limited range of artifacts, namely vehicles, and for locations.

Mr. Sears was invited to address the crowd. But the reticent program manager declined.

<ENTITY-1> :=
ENT_NAME: "Sears"
ENT_DESCRIPTOR: "the reticent program manager"
ENT_TYPE: PERSON
ENT_CATEGORY: PER_CIV

Figure 2. A Template Element

Template Relation

The Template Relation task currently captures three types of relations in the text:

- the *location-of* relation ties an organization to one of its locations
- the *employee-of* relation ties a person to the organization he/she works for
- the *product-of* relation ties an artifact to the organization that produced it

For example, from the text

> Capt. Dennis Gillespie was the commander of the Navy's Fighter Squadron 213, based in Miramar Naval Base near San Diego.

one should capture the relations that Capt. Dennis Gillespie is employed by the Navy, and that Fighter Squadron 213 is located in Miramar Naval Base. Template relations are recorded as relations between, or properties of, template elements.

Scenario Template

A *scenario*, in MUC terminology, is a description of a class of events. Examples of scenarios which have been used in prior MUCs include naval sightings and attacks, terrorist events, formation of international joint ventures, union negotiations, management succession (corporate executive hirings and firings), and plane crashes. Each scenario includes a list of the pieces of information to be provided about the event. For management succession, for example, the information included the identity of the person and the corporation, the position at the corporation, the reason for starting or leaving the job (if given), etc.

The information about the event is used to fill a template for the scenario. Some slots in the template are filled with strings, while others are filled with pointers to lower level objects (template elements and other objects), creating a hierarchical structure of information about each event. Figure 3 shows what a filled management succession scenario template might look like. This is considerably simplified from the actual MUC-6 template, and uses MUC-7-style template elements.

Richard Relish will retire next week as president of the famous fast food restaurant, Frank's Franks Inc.

<EVENT-1> :=
COMPANY: <ENTITY-2>
PERSON: <ENTITY-1>
POSITION: "president"
STATUS: OUT

<ENTITY-1> :=
ENT_NAME: "Richard Relish"
ENT_TYPE: PERSON
ENT_CATEGORY: PER_CIV

<ENTITY-2> :=
ENT_NAME: "Frank's Franks"
ENT_TYPE: ORGANIZATION
ENT_CATEGORY: ORG_CO

Figure 3. A filled Scenario Template.

The scenario template task changes for each evaluation. The other tasks are expected to remain roughly constant from one evaluation to the next, although we may be including additional relationships in the template relations task.

Some comparisons

To conclude this summary of extraction tasks, we can contrast them with other types of tasks. We can distinguish document retrieval (or "information retrieval") from information extraction by noting that document retrieval normally retrieves an entire document (or, possibly, a passage) in response to a request for information, while information extraction retrieves very selective information from the document — "just the facts".

Full text understanding has the goal of identifying and formalizing *all* the information in a text. This is, of course, a very open-ended problem. In contrast, in information extraction we specify in advance (in the scenario specification) the types of objects and relationships which we wish to capture from the text. This limits the problem and defines the output structure. Further, by creating a well-defined target it makes it possible to evaluate systems against one another.

THE STRUCTURE OF INFORMATION EXTRACTION SYSTEMS

Alternative approaches to extraction

For the named entity task, the general approach has been to write a set of finite-state (regular) patterns to capture the different types of entities. For newspaper text, the craft of writing these patterns is highly developed; performance of the best named entity taggers is about F=0.96, comparable to human performance on this task (the F measure is de-

scribed on the next page). Because of the simplicity of the task, there have been a number of efforts recently to learn named entity rules from tagged corpora, using techniques such as stochastic models based on n-gram statistics (Bikel et al. 97) and decision trees.

For the remainder of this section, we will focus on systems for the scenario template task. In many implementations, the more basic template annotations (template elements and template relations) are produced "along the way" to creating the scenario templates.

Although there are many different extraction systems, the overall structure of most of these systems is quite similar. The most evident difference is in the degree of syntactic analysis which is performed. Some systems perform a full syntactic analysis: each input sentence is parsed, and then the syntactic (tree) structures are analyzed for instances of particular (semantic) patterns related to the events of interest. This approach is limited by the accuracy of full-sentence syntactic analyzers. While the performance of these analyzers has been steadily increasing of late, the limitations of these parsers have led groups to pursue alternate approaches, using either partial parsing or no syntactically-based analysis at all.

Many of these systems employ what is often referred to as "pattern matching": bottom-up deterministic analysis performed in several stages, with each stage using a set of finite-state patterns. In some systems, all of these stages involve semantically specific patterns (that is, patterns which refer to semantic word and phrase classes). Other systems combine semantically based and syntactically based patterns.

The particular extraction system we shall describe here, the NYU Proteus system (Grishman 95), follows this middle road. It uses a limited degree of syntactic analysis, for noun groups and verb groups, along with semantically specific clause-level patterns.[3]

The Proteus Extraction System

In the Proteus extraction system, processing of each sentence begins with lexicon look-up (using a large English syntactic dictionary augmented by lists of names and specialized terms) and part-of-speech tagging using a statistical tagger.[4] This is followed by a series of pattern-matching stages, which identify and mark successively larger constituents. The main stages are

- name patterns
- noun group and verb group patterns
- noun phrase patterns
- clause (event-specific) patterns

Each stage consists of a set of patterns; each pattern is a regular expression, whose elements may be literals, lexical categories (defined in the dictionary), or constituents defined by an earlier set of patterns. For each set of patterns, the sentence is processed from left to right. Starting at each token, the system attempts to match each pattern in the set against the remainder of the sentence. If the match is successful, an associated action is performed, normally the creation of a new constituent. If several patterns match, or one pattern matches in several ways, the longest match is taken (and, among matches of equal length, the first is taken). These successive stages of pattern matching thus perform a deterministic, bottom-up partial analysis of the sentence.

Pattern elements may also include semantic class constraints, stated in terms of a task-specific semantic classification hierarchy. The noun group and verb group patterns are general patterns stated in terms of syntactic classes alone. On the other hand, most of the noun phrase patterns, and all of the clause patterns, are domain or task-specific, and make reference to these semantic classes.

For example, one of the clause-level patterns for events where a person is appointed to a position is

```
np(C-organization) sa vg(C-appoint) sa
        np(C-person) sa "as" np(C-position)
```

This matches examples such as "IBM yesterday appointed Fred Smith as president." C-organization, C-appoint, C-person, and C-position are semantic class constraints. "sa" stands for "sentence adjunct", and allows for intervening modifiers such as "yesterday".

The noun phrase and clause patterns create a "logical form": an internal representation of the events and entities of interest for the scenario. This logical form is then processed by a reference resolution component, which identifies references to previously mentioned entities and events. Finally, when the entire article has been processed, some simple, task-specific inferencing is done and the logical form is mapped into the required template structure.

EVALUATION OF EXTRACTION PERFORMANCE

As we noted earlier, a centerpiece of the Message Understanding Conferences has been the comparative evaluation of the participants' systems on a variety of extraction tasks. Some time before the evaluation, the conference organizers provide a specification of the scenario and training data for the various tasks. For the evaluation they select additional messages or articles and create an "answer key" — hand-prepared data base entries indicating what information *should* be extracted from the text.

The answer key makes it possible to evaluate the performance of an extraction system — to see how closely its output matches the key. The two most common measures of

---

[3] In this regard we have followed the lead of SRI International and their FASTUS information extraction system (Appelt et al. 93).

[4] We wish to thank BBN for providing their part-of-speech tagger, POST.

performance are recall and precision. Roughly speaking, for the template tasks,

$$\text{recall} = \frac{\text{number of slots correctly filled}}{\text{number of filled slots in key}}$$

$$\text{precision} = \frac{\text{number of slots correctly filled}}{\text{number of slots filled by system}}$$

In addition, a composite F score is generally reported:

$$\text{F} = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$$

EXTRACTION ON SPEECH TRANSCRIPTS

As we have seen, extraction systems typically operate by looking for certain structures — sequences of constituents — in the input text, either through full (syntactic) parsing or through more limited, semantically-directed pattern matching. Such methods work effectively for well-edited, relatively error-free texts such as on-line newspapers. However, their performance degrades rapidly when applied to texts with a significant number of errors, such as those produced by speech recognition. (Weischedel et al. 96) briefly discuss the effect of transcription errors on the Template Element task.

We consider here how to reduce this degradation in performance, again focussing on the Scenario Template task: how to extract at least partial information from text which includes a substantial number of input errors.

Adapting to Speech Errors: Modifying the Patterns

The patterns which have been developed for the information extraction system are designed to account for the variety of syntactic structures, including active clauses, passive clauses, relative clauses, conjunction, etc. However, patterns such as the example above, with several required elements, are quite sensitive to errors in the input text. If any of the required elements are missing, or an extraneous token intervenes between the elements, the pattern won't match. As the error rate increases to the point where 20% to 30% of the tokens are incorrect, the extraction performance plummets to near 0. However, intuitively, looking at the errorful text, we can very often identify sequences which are corrupted versions of events of interest. For example, even in the text

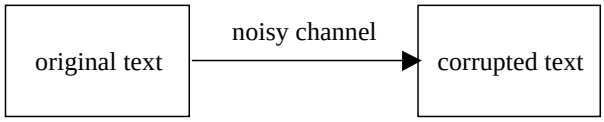
```
HE'S EXCEED LANCE R PROMISE COUPS SEPTEMBER WAS NAMED PRESIDENT AND CHIEF AGREEING OFF SERVED PARENT
```

which is a low-quality transcription of

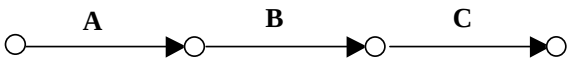
```
HE SUCCEEDS LANCE R. PRIMIS WHO IN SEPTEMBER WAS NAMED PRESIDENT AND CHIEF OPERATING OFFICER OF THE PARENT
```

we can discern that the text is talking about an appointment as president and some additional position, even though we can't get all the details (the fellow's last name and his second position) quite right. How can we adapt our extraction system to make similar identifications?

Consider a simple model in which the errorful text is created by sending the original text through a noisy channel which both deletes some incoming tokens and inserts random tokens (drawn from some set of tokens **Y**) into the text.[5]

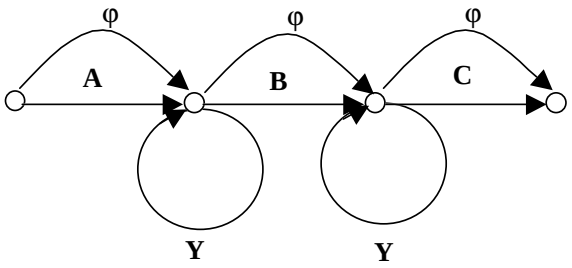


Further, let us suppose for simplicity that the extraction system is just looking for a single pattern, **A B C**. We can represent this pattern by the graph:



If we built a recognizer based on this graph, we would identify an event each time arc **C** was traversed.

Suppose now we wish to construct a corresponding graph representing the corrupted text. What would this graph look like? To account for the possible insertion of tokens drawn from distribution **Y** after each token of the original text, we introduce loops labeled by **Y** into the graph. To account for the possible deletion of input tokens, we introduce a null arc (labeled by φ) into the graph, in parallel with each original arc. We end up with the graph



In other words, we need to modify the original pattern to skip pattern elements (the null arcs) or to skip input tokens (the **Y** arcs).

EXPERIMENTS

Preparation of speech transcripts

[5] We model a substitution as a deletion followed by an insertion.

To evaluate our approach, we used the "management succession" scenario developed for MUC-6 (MUC 95, Grishman and Sundheim 96), which involves information about people who enter or leave executive positions in companies.

From the 100 training articles provided for MUC-6, we selected 11 which included a large number of succession events (altogether, 47 instances of starting or leaving a position). These articles were read aloud and recorded by a member of our staff. The recordings were segmented into utterances using a simple program for detecting periods of relative silence. The utterances were then transcribed using the SPHINX-II speech recognition system (Huang 93) using a language model for business news articles.

We did adjust several parameters, including in particular the segmentation criteria, in order to generate different qualities of transcripts. However, since we were specifically interested in generating transcripts with substantial numbers of errors, no great effort was made to optimize the transcription quality; the transcripts should not be taken as representative of SPHINX recognition quality for a well-tuned system.

Adapting the patterns

We have modified our extraction system, first to handle an idealized speech transcript, and then to handle the errors introduced by the speech recognizer.

The modifications for an idealized speech transcript were quite simple. We removed all punctuation from the extraction patterns. We allowed for spelled-out numbers ("nineteen ninety eight") in places where digit sequences had been expected. Finally, we modified our name recognizer, which normally uses case information to detect names (most names are capitalized), to rely instead on the part-of-speech tagger to distinguish names from common nouns.

To handle real, errorful speech transcripts, we basically followed the approach outlined above. We modified the pattern matching engine in our extraction system so that individual pattern elements would be optional (corresponding to the null arcs in the graph), and to allow one or more tokens to be skipped between pattern elements. Things are not entirely straightforward, however, for several reasons:

First, the modified patterns described above will be effective only if it is unlikely that the tokens inserted by our noisy channel will match elements sought in our pattern. This will be the case, generally speaking, for the semantically-specific patterns in the noun phrase and clause pattern sets; the chances are quite small that randomly inserted tokens will constitute a verb group of semantic class *appoint* or a noun phrase describing a position. [6] This is less true for the "low-level" patterns for names, noun groups, and verb groups. The chances are quite large that a randomly inserted token will be a valid part of a pattern for a company name, a noun group, or a verb group. For example, almost any adjective or singular noun (a large fraction of the vocabulary) would match a noun group pattern following a determiner.

We have therefore applied the pattern modification only to the patterns in the noun phrase and clause pattern sets. In terms of our noisy channel model, we view the input as a sequence of noun groups and individual tokens (outside of noun groups); the channel may insert or delete a noun group or an individual token outside of a noun group.

Second, the modified graphs will typically allow several different analyses of an input. Suppose we have patterns ABC and AC, and we have input ABC. This input could match the ABC pattern (no tokens skipped), or the AC pattern (skipping token B). In this case, we will prefer the match against the ABC pattern, skipping no tokens. More generally, if several matches are possible (with different numbers of skipped tokens), the system selects the match for which the *maximum* number of elements skipped between any pair of elements in a pattern is minimal. We place a limit on the maximum number of tokens skipped between any pair of pattern elements. For our test corpora, we established a limit of 5 tokens; we determined experimentally that performance did not improve beyond 5 skipped tokens.

Third, in allowing for deletions we have restricted the pattern elements which can be deleted. In particular, for clausal patterns, we have required that the verb group be present. Without such a constraint, we would get a large number of false matches.

In many cases, the text which matches a pattern element ends up filling a slot in the output data base. If we hypothesize that the text was deleted, we must decide what to put in the data base slot. The simplest possibility, of course, is to leave the slot blank. However, we have also experimented with another possibility, in which we attempt to recover the element from prior discourse using standard reference resolution methods. For example, if the company name is omitted, the system will search for the most recent prior element of type *company*.

Results

In order to observe the efficacy of our model over a range of error rates, we have used two different transcripts, produced from the same audio file with different settings for some sentence segmentation and recognition parameters. The insertion, deletion, and combined word error rates for the two transcripts are:

	insertion rate	deletion rate	word error rate
transcript 1	16.0%	13.7%	19.5%
transcript 2	25.7%	24.1%	32.1%

In addition, we ran our base (MUC-6) system on the origi-

[6] This is not as true for people's names; randomly generated tokens do occasionally get tagged as a person's name.

nal (mixed-case) text, and on a "perfect transcript", all upper case text with punctuation removed. The results for these different combinations of transcripts and patterns are as follows:

corpus	patterns	recall	precision	F
mixed case	original	62%	74%	0.68
perfect transcript	original	36%	70%	0.48
transcript 1	original	15%	73%	0.26
transcript 1	modified	23%	76%	0.35
transcript 2	original	5%	85%	0.10
transcript 2	modified	17%	75%	0.28

We note that there is a significant loss in recall in moving from the mixed case text to the perfect transcript. This reflects the fact that our name recognizer was tuned for mixed-case text, and has been only minimally modified to work with the monocase transcript. However, the fall-off in performance in even greater when we move to the real transcripts; for transcript 2, with 30% word error, we lost over 85% in recall compared to the perfect transcript. While recall on such a heavily corrupted text is necessarily still poor with our modified patterns, the improvement is striking (about a factor of 3).

LOOKING AHEAD

We have shown that it is possible, though quite simple methods, to recover a substantial portion of the loss of performance of an information extraction system when provided with moderately noisy input data. Such a system could have application, for example, in performing extraction on broadcast news data, where recognition with word error rates of 20% to 30% is now possible. In this closing section, we speculate on a number of areas of inquiry which might result in substantial additional gains in performance.

Name Recognition

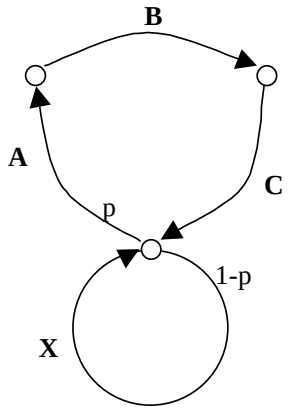
A substantial part of our loss in performance reflects the problems of our system in recognizing names. This is really a combination of two types of problems.

First, even when the individual tokens are transcribed correctly, our system — which has been crafted primarily for mixed-case text — does a poor job of finding the names. This is reflected in the considerable drop in performance when moving from the mixed-case text to the perfect transcript. It should be possible to eliminate much of this drop using a name recognizer better tuned to monocase text. In particular, BBN has recently demonstrated a trainable name recognizer based on n-gram statistics whose performance on monocase text is close to that of the best systems on mixed-case text (Bikel et al. 97), and whose performance on speech transcripts is only slightly worse (Kubala et al 98).

Second, it is well known that correctly transcribing proper names is a difficult task. (Weischedel et al. 96) discuss this in the context of the Template Element task, and suggest several alternatives. Among the possibilities they mention are greatly enlarging the vocabulary to include infrequent proper names (improving proper name recognition at some cost in overall accuracy) and tagging out-of-vocabulary (OOV) tokens for subsequent manual transcription. They note that the latter is not very promising for the Template Element task; one would probably have to transcribe all the OOV items to produce a useful output. For the Scenario Template task, on the other hand, it should be sufficient to transcribe only OOV items which fit into scenario patterns; in some applications, one might record the audio signal as part of the template, and wait until the Scenario Templates are used in some retrieval or analysis task before transcribing items of potential interest.

Probabilistic Models

The mechanisms described above have attempted to use some intuitive notion of preference to select among alternative pattern matches — for example, to prefer the match which skips the minimal number of tokens. It should be possible to provide a firmer basis for these preferences through a probabilistic model. We can observe the relative frequencies of the different patterns in the original text, and use this to create a probabilistic model of the original text for this scenario. This model would include probabilities for tokens which fall outside the patterns, perhaps using unigram probabilities. For example, if we had our pattern **ABC** (occurring with probability p) and a background distribution of tokens **X**, our model would look like



Next, one would measure insertion and deletion rates at the level of names and noun phrases, and use these to create a statistical model of the corrupted text, like the model we created before. A probabilistic pattern matching could then be performed, seeking the most probable path through this graph. An examination of the results of our current pattern matching suggests that this approach could produce some improvements in performance.

The probabilistic approach should also allow us to obtain some recall/precision trade-offs more directly than with our

current, deterministic system. For example, raising the probabilities of the patterns in the basic (uncorrupted text) model could be used to increase recall. In addition, it may be possible to increase recall by searching the word lattice produced by the recognizer, rather than using just the top hypothesis from the recognizer.

Dealing with multiple pattern sets

The current extraction architecture makes deterministic decisions at each stage of pattern matching. This has proven generally satisfactory for text applications. For processing speech transcripts, however, where the uncertainties are greater, it may be better to propagate a limited set of alternative analyses from one stage to the next, as a lattice with probabilities marked. This seems natural, for example, if we use a corpus-trained name tagger which makes probabilistically based decisions in any case, and currently only outputs the top-ranking choice.

We may also obtain some improvement in performance by replacing the current syntactic patterns for noun groups with semantically constrained patterns. While the syntactic patterns are effective for text analysis, they are easily led astray by noise in the speech applications, since a large fraction of inserted words (nouns, adjectives) would match an element of a noun group. Semantically constrained noun group patterns would inevitably lead to some loss of coverage, but would be more robust against the noise of randomly inserted tokens.

ACKNOWLEDGEMENT

The work reported here was supported in part by the Defense Advanced Research Projects Agency under contract DABT63-93-C-0058 from the Department of the Army.

REFERENCES

(Appelt et al. 93) D. Appelt, J. Hobbs, J. Bear, D. Israel, and M. Tyson. Fastus: a Finite state processor for information extraction from real world text. In *Proc. 13th IJCAI*, Chambery, France, August 1993.

(Bikel et al. 97) Daniel Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: A High-performance Learning Name Finder. *Proc. Fifth Conf. on Applied Natural Language Processing*, 1997, pp. 194-201.

(Grishman 95) Ralph Grishman, Where's the Syntax? The NYU MUC-6 System. *Proc. Sixth Message Understanding Conf. (MUC-6)*, Nov. 1995. Morgan Kaufmann.

(Grishman and Sundheim 96) Ralph Grishman and Beth Sundheim, Message Understanding Conference-6: A Brief History. *Proc. COLING 96.*

(Huang 93) X. Huang, F. Alleva, H. Hon, M. Hwang, K Lee, and R. Rosenfeld. The SPHINX-II Speech Recognition System: An Overview. *Computer Speech and Language* **2** (1993), pp. 137-148.

(Kubala 98) Francis Kubala, Richard Schwartz, Rebecca Stone, and Ralph Weischedel. Named Entity Extraction from Speech. This volume.

(MUC 95) *Proc. Sixth Message Understanding Conf. (MUC-6)*, Nov. 1995. Morgan Kaufmann.

(Weischedel et al. 96) Ralph Weischedel, Sean Boisen, Daniel Bikel, Robert Bobrow, Michael Crystal, William Ferbuson, Allan Wechsler, and the PLUM Research Group. Progress in Information Extraction. *Advances in Text Processing: Tipster Program Phase II*. 1996.